Teaching ICT Quality

Stephen B. Seidman
Texas State University

What <u>is</u> quality?

- The Oxford English Dictionary gives 16 primary definitions for “quality”.
- Definition 8.c is most relevant to the theme of this conference. In this long-standing usage (the OED gives a citation from 1771), “quality” is defined simply as “excellence, superiority”.

Excellence, superiority?

- The importance of context
  - Excellence, according to whom?
  - Superiority, compared with what?

Quality in ICT

- Topics from the conference call for papers
  - Quality in Agile Methods
  - Quality in Web Engineering
  - Quality Evolution
  - Quality in Verification and Validation

Dimensions of ICT quality

- Algorithm quality
  - Design: elegance?
  - Implementation: performance, metrics, scalability
- Architecture quality
  - Hardware or software?
  - Elegance?

- Software quality
  - Processes or products?
  - “the degree to which a set of inherent characteristics fulfills requirements” (ISO 9001)
  - Functionality, reliability, usability, efficiency, maintainability, portability (ISO 9126)
  - Three levels of quality (Denning 1992)
    - All basic promises were met.
    - No negative consequences were produced.
    - The customer was delighted.

- User interface quality
  - (Raskin, *The Humane Interface*, 2000)
    - An interface should be
      - effective, habituating, reliable, efficient, and tested

What can we do with these ideas?

- How do we assess quality on any of these dimensions?
- How do we teach our students to seek and assess quality?

Assessing ICT quality

- Each dimension has its own approach to assessing quality
  - Quantitative approaches: metrics
  - Qualitative approaches: the “-ilities”

Lessons from engineering?

- The engineering profession is much older than ICT and equally interested in quality.
- How does engineering think of, assess, and teach quality?
- Are there lessons to be learned from engineering?

What is the engineer’s ultimate goal?

- A manufacturing process to produce large numbers of products?
- A single product?
  - bridge, power plant, refinery, ...

Quality engineering for manufacturing

- Product quality
    - effectiveness (suitability for purpose) of the products produced
- Design quality
    - originality, feasibility
- Process quality
        - TQM: process quality as conformance to internal requirements
        - Six Sigma: process quality as defect reduction via data/statistical analysis

Teaching about process quality

- This topic is always part of industrial, systems, and manufacturing engineering curricula; it sometimes appears in other engineering curricula.
- The following image describes a “six-sigma” course from MIT.

Students working together on an implementation exercise. (Photograph by Hugh McManus.)

Hugh McManus

Al Haggerty

Prof. Annalisa Weigel

Course Features
Course Description
Technical Requirements

Course Features

Video lectures

Course Description

This course introduces the fundamental Lean Six Sigma principles that underlay modern continuous improvement approaches for industry, government and other organizations. Lean emerged from the Japanese automotive industry, particularly Toyota, and is focused on the creation of value through the relentless elimination of waste. Six Sigma is a quality system developed at Motorola which focuses on elimination of variation from all processes. The basic principles have been applied to a wide range of organizations and sectors to improve quality, productivity, customer satisfaction, employee satisfaction, time-to-market and financial performance.

This course is offered during the Independent Activities Period (IAP), which is a special 4-week term at MIT that runs from the first week of January until the end of the month.

Technical Requirements

Special software is required to use some of the files in this course: .xls.

- There are many graduate and continuing education programs and courses in “quality engineering” that emphasize process quality and statistical techniques.
- For example, Rutgers University (New Jersey, USA) offers a professional continuing education course titled “Fundamentals of Quality Engineering”:

Fundamentals of Quality Engineering

PDH hours available

REGISTER

Program Overview

The Fundamentals of Quality Engineering is an introductory course that addresses issues of quantifying quality, use of quality control approaches for variables and attributes, shift detection in process mean and variance, optimum tolerance design, quality loss and multivariate quality control. Quality engineers are responsible for ensuring the highest standards of products and services. These specialized engineers maintain these standards by identifying mistakes and learning to prevent them.

Who Should Attend?

This course is for professional engineers with an interest in understanding the fundamentals of product quality and control.

Skills You Will Acquire

You will gain an understanding of the quality control system. You will also learn how to evaluate and prevent product failures, how to work with control charts, and tolerance design.

Program Outline

1. Quality Value and Engineering
2. Quality Definitions
3. Quality Engineering
4. Quality and Process Capability
5. Traditional Quality Control Process
6. Variable Control Charts
7. Attribute Control Charts
8. Loss Function and Quality Level
9. Derivation of the Loss Function
10. Types of Characteristics
11. Quality Evaluations for Different Characteristics
12. Tolerance Design
13. N Type
14. L Type
15. S Type

Quality engineering for single products

- Process or product quality?
  - What is the quality of a refinery?
    - Is it the same as the quality of the production process that the refinery embodies? What about the quality of the design process, or of the design?
    - Avoidance of failures?
      - “Failure is an unaccepted difference between expected and observed performance” (ASCE)

- What is the quality of a bridge?
  - Conformance to requirements
  - Maintainability?
  - Avoidance of failures?
  - Quality of the design process, or of the design?

Teaching about quality via failure

- Engineers have always learned from failure:
  - “The single most fruitful source of lessons in engineering judgment exists in the case histories of failures, which point incontrovertibly to examples of bad judgement and therefore provide guideposts for negotiating around the pitfalls in the design process.” (Petroski, *Design Paradigms*)

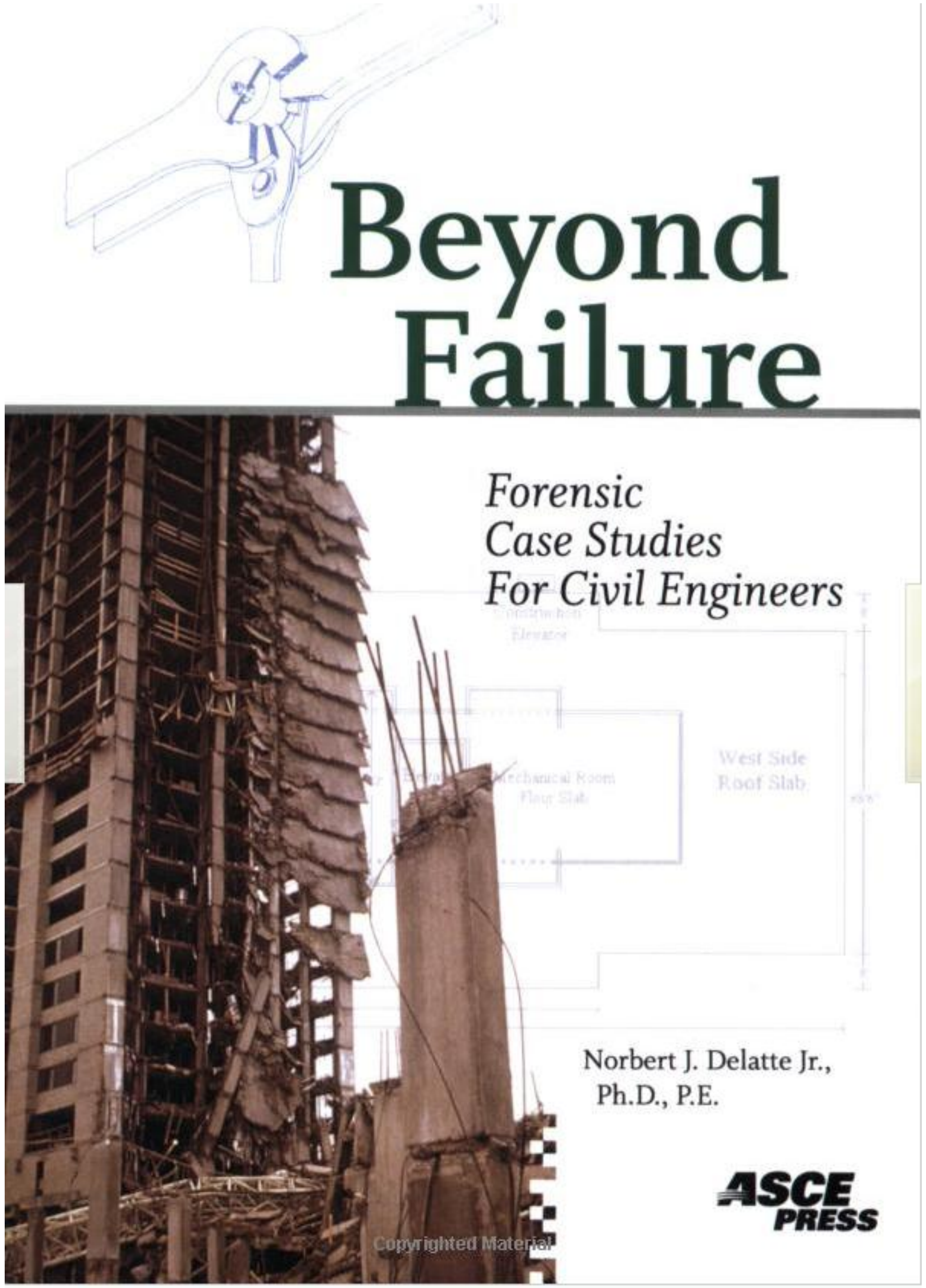
- Success is a much less successful teacher
    - “Basing structural extrapolation on models of success rather than on failure avoidance ... [culminated] in the collapse of the Tacoma Narrows Bridge in 1940.” (Petroski, *Design Paradigms*)

- Failures have had an important impact on engineering practice.
- Teaching engineering students about quality therefore requires that we teach them about failure.

Failure in the engineering curriculum

- MIT: Design by Failure: proposed course for all first-year engineering students
  - Examples from all engineering disciplines
    - Civil: Tacoma-Narrows Bridge failure
    - Chemical: Bhopal accident
    - Nuclear: Chernobyl

- Rochester Institute of Technology
  - Use of aircraft crash cases in teaching
    - design failures
    - maintenance failures
    - humans and computers in the loop



Beyond Failure

Forensic Case Studies For Civil Engineers

Norbert J. Delatte Jr., Ph.D., P.E.

ASCE PRESS

FOREWORD xi
by Kenneth L. Carper

PREFACE xiii

1 **Why Case Studies?** 1
Organization of This Book 3
Notes to the Student 4
Sources for Case Materials 5

2 **Statics and Dynamics** 7
Hyatt Regency Walkway 8
Tacoma Narrows Bridge 26
Aircraft Impacts 38
Other Cases 42

3 **Mechanics of Materials** 51
Quebec Bridge 51
Point Pleasant Bridge Collapse 70
Comet Jet Aircraft Crashes 82
Other Cases 86

4 **Structural Analysis** 89
Agricultural Product Warehouse Failures 90
Ronan Point 97

L'Ambiance Plaza Collapse 107
Cleveland Lift-Slab Parking Garage 121
Kemper Arena 124
Other Cases 126

5 **Reinforced Concrete Structures** 129
Air Force Warehouse Shear Failures 130
2000 Commonwealth Avenue 133
Skyline Plaza in Bailey's Crossroads 144
Harbour Cay Condominium 149
Bombing of the Oklahoma City Murrah Federal Building 155
The Pentagon Attack 162
Other Cases 167

6 **Steel Structures** 173
Hartford Civic Center Stadium Collapse 174
Mianus River Bridge Collapse 184
Cold-Formed Steel Beam Construction Failure 188
The World Trade Center Attacks 195
Pittsburgh Convention Center Expansion Joint Failure 206
Minneapolis I-35W Bridge Collapse 211
Other Cases 215

7 **Soil Mechanics, Geotechnical Engineering, and Foundations** 221
Teton Dam 223
Vaiont Dam Reservoir Slope Stability Failure 234
The Transcona and Fargo Grain Elevators 249
Other Cases 255

8 **Fluid Mechanics and Hydraulics** 257
Johnstown Flood 257
Malpasset Dam 267
Schoharie Creek Bridge 277
New Orleans Hurricane Katrina Levee Failures 287
Other Cases 299

9 **Construction Materials** 301
Austin Concrete Dam Failure 303
Liberty Ship Hull Failures 310
Willow Island Cooling Tower Collapse 316
Boston's Big Dig Tunnel Collapse 325
High-Alumina Cement 330
Other Cases 330

10 **Management, Ethics, and Professional Issues** 333
Citicorp Tower 333
Space Shuttle *Challenger* 345
Sampoong Superstore, Korea 352
Misuse of the Professional Engineer License 357
Property Loss Investigations 359

APPENDIX A: Notes to the Professor 361

APPENDIX B: The ASCE Code of Ethics 377

APPENDIX C: Some Cases on Video and DVD 385

REFERENCES 391

INDEX 403

ABOUT THE AUTHOR 407

Teaching ICT Students about Quality

- Software engineering
  - For many years, software engineering (and software engineering education) has paid explicit attention to “quality”.
- ICT Job frameworks
  - Quality topics appear explicitly in European job frameworks.
- Computer Science
  - How do we teach beginning students about code and algorithm quality?

Quality in Software Engineering

- SWEBOK (IEEE-CS body of knowledge)
  - Fundamentals
    - Value and cost of quality
  - Static techniques
    - Formal methods
    - Verification and validation
    - Reviews and audits
  - Dynamic techniques
    - Testing

- Processes vs. Products
  - Process quality
    - Standards for software engineering processes
    - “... the quality of a software product is largely determined by the quality of the software development and maintenance processes used to build it” (Paulk, 1995)
  - Product quality
    - applies to all software products: requirements specifications, architectures, designs, code, test plans, documentation, reports, ...

- SE2004
    - Software quality is a pervasive concept that affects, and is affected by, all aspects of software development, support, revision, and maintenance.
    - It encompasses the quality of work products developed and/or modified (both intermediate and deliverable work products) and the quality of the work processes used to develop and/or modify the work products.
    - Quality work product attributes include functionality, usability, reliability, safety, security, maintainability, portability, efficiency, performance, and availability.

- Software quality concepts and culture
- Software product quality standards: ISO-IEC 9126
- Software process quality: ISO 9000, CMMI
- Software quality processes: IEEE 730, IEEE 1061
- Process assurance: planning and reporting
- Product assurance
  - Root cause analysis, defect prevention
  - Metrics and measurement
  - Assessment of quality attributes

Teaching SW engineers about quality

- The usual approach
  - Faculty lectures on quality topics and standards
  - Student readings of appropriate materials
  - Problems with this approach
    - Lack of student interaction
    - Disconnection with project work

Other approaches

- Problem-based learning (PBL):
  - a student-centered instructional strategy in which students collaboratively solve problems and reflect on their experiences (Wikipedia)
  - Characteristics include
    - Learning is driven by challenging, open-ended, ill-defined and ill-structured, practical problems.
    - Students work in collaborative groups.
    - Teachers take on the role of "facilitators" of learning.

- (Richardson and Delaney, 2010) Teams of M.Sc. students in a SwE quality module at the University of Limerick were asked to write the software quality plan for a hospital. They first viewed a video that gave them a sense for the context:
  - www.youtube.com/watch?v=-xrrk-XhgVc

- The lecturer served as a facilitator to the groups, circulating among them.
- Short lectures were given on specific topics, as needed.
- A subject-matter expert was made available to the groups.

- Authors’ conclusions:
  - PBL seems to have significantly increased student involvement and satisfaction.
  - PBL may not suit all student learning styles.
  - Assessment of student learning may be more difficult with PBL.
  - It’s important to have strong links with industry when using PBL to teach software quality.

- Major project course
  - (Towhidnejad 2002; Doerschuk 2004)
    - Example 1: This undergraduate class is structured as a small software development organization. divided into five-member development teams; students use Humphrey’s PSP.
      - Build 1: generation of requirements and design documents and a prototype
      - Build 2: development of final product

- The software quality assurance function is provided by graduate students in a software testing course.
  - Build 1
    - formal inspection of requirements and design specification documents
    - construction of test plan from requirements specification
  - Build 2
    - undergraduate students provided internal SQA functions
    - graduate students provided external SQA functions, such as independent verification and validation

- Example 2: Student development teams are divided into subteams
  - Team A develops the design
  - Team B inspects the design
- Teams switch roles between development phases

- Comments
  - The authors looked at interaction between students, but this could have been studied more formally (phenomenography?)
  - Both papers observed that undergraduate students don’t seem to understand the importance of software quality.
  - It’s not easy to incorporate realistic software processes (including quality processes) into student environments.

The need to teach about failure

- It’s as important to expose software engineering students to failures as it is in other branches of engineering.
- System and software engineering failures are well-known and widely reported.

- Well-documented major cases include
  - London Ambulance Service dispatching system (1987-92)
  - THERAC-25 radiation therapy instrument (1985-87)
  - US Federal Aviation Administration (FAA): Advanced Automation System ($2.5 billion, 1989-1994)
  - US Federal Bureau of Investigation (FBI): Virtual Case File ($170 million, 2001-2005)

- Failures do appear in computing curricula; the THERAC-25 is often discussed in courses dealing with ethics and technology.
- Larger software engineering failures can be considered as examples of failed software engineering processes:
  - FAA: incomplete and unstable requirements
  - LAS: testing; verification and validation

- However, software engineering students are rarely if ever presented with detailed case studies of system/software failures.
- Few complete case studies are available. Exceptions include:
  - LAS: www.cs.ucl.ac.uk/staff/A.Finkelstein/las/lascase0.9.pdf
  - Therac-25: Leveson, *Safeware, System Safety and Computers*, 1995

- However, the study of failure needs to play a role across the software engineering curriculum.
- Examples of industrial-strength design and architecture failures would be particularly interesting.
  - Parnas’ 1972 paper *On the Criteria to Be Used in Decomposing Systems into Modules* is useful in this context, but the application is still rather small.

Quality in ICT Job Frameworks

- Job frameworks for ICT
  - SFIA (UK), AITT (Germany), CIGREF (France)
  - European e-Competence reference framework (www.ecompetences.eu)
    - Topics listed include *quality strategy development, quality assurance* and *quality management*

- ICT Lane: qualifications framework, within the ambit of the European Qualifications Framework
  - The idea is to link qualifications to appropriate framework items.
  - It would therefore be possible in principle to identify a specific qualification that would correspond to a quality item in a job framework.
  - We could then see how ICT quality topics are taught to those seeking this qualification.
  - However, all of this is still a work in progress.

A Framework for ICT Foundation Degrees

- A recent effort in the UK (Foundation Degrees Forward: www.fdf.ac.uk) is explicitly linked to the SFIA framework.
- A degree specification was developed as part of a strategic partnership between ICT employers and higher education institutions.

- One of the FDF learning outcomes is devoted entirely to quality; its indicative content includes
  - The components of software quality – internal and external aspects; validation and verification, reliability, conformance, completeness, maintainability
- This specification requires courses to be collaborative efforts between industry and academia.
- Workplace efforts and assessment are therefore an essential part of such courses.

- Although the FDF specification of foundation ICT degrees appears to be well-founded, it remains to be seen how quality topics are actually taught/learned and assessed within this hybrid university/workplace educational model.

The need to teach ICT students about code quality

- What do our students mean when they talk about the quality of a computer program?
- Where do they get their ideas about code quality?
- How can we teach introductory students about code quality?

- A recent study (Lewis et al, CACM, May 2010) surveyed student opinions on the following statements:
  - *On a programming assignment, what matters is getting the desired result, not how you arrive at the result.*
    - 58% of first-year students agreed
    - 54% of final-year students agreed

- *If a program works it doesn’t matter how it is written.*
    - 45% of first-year students agreed
    - 20% of final-year students agreed

- *Doing things the “right” way isn’t as important as just pushing through to a solution.*
    - 56% of first-year students agreed
    - 32% of final-year students agreed

- This data was obtained from CS students at a major US university. Although quality was not the specific target of this study, it suggests that a significant proportion of successful CS students may lack a strong sense for the meaning of code quality.

- This topic is pursued further in a recent paper by Pears. He surveys literature on student learning in programming courses, and reports that:
  - Early learning in programming is dominated by syntactic concerns.
  - Most students lack a holistic view of program function.
  - Students focus on code at the operation/line level.

- He summarizes this literature by saying
  - “...many students at the conclusion of an introductory programming course when given a functional description are unable to write a piece of software that meets the requirements.”
  - “...many students are not able to explain what a piece of code does in a more advanced manner than the line by line approach ... ”

- Kolikant and Mussai (2008) state
  - "... as long as they had any operations written correctly students considered the program partially correct"
  - They feel that this conception may derive from the way in which student programs are assessed.

- Students often feel that if a program compiles it is correct (and therefore of good quality).
  - ... there were some students who [at the end of the academic year] still believed ... that a program is correct when it is free from syntactical errors” (Stamouli and Huggard, 2006)

What can be done?

- Some scholars (e.g., Patton and McGill, 2006) propose that software quality can be taught by utilizing longitudinal portfolios of software artifacts, along with automated software quality metrics.

- Pears suggests that educators should
  - adopt an approach to code development based on holistic goals and plans
  - design formative assessment strategies that emphasize quality aspects
  - include testing and debugging as explicit parts of the curriculum

Build on innate student understanding

- A recent paper (CACM, July 2010), examined the concepts of concurrency held by students at the start of an introductory programming course.
- It concluded that these students' intuitive understanding of concurrency was roughly equivalent to that of experienced students beginning a concurrency course.

- If this is the case for concurrency, perhaps students also have intuitive concepts of software quality. (Future research topic)
- If this is the case, can such concepts be used as the basis for a deeper understanding of software quality?

Conclusions

- How should we teach ICT students about quality?
- It depends on the target; are we interested in <u>process</u> quality or <u>product</u> quality?

- Process quality
  - It may make sense to teach engineering process quality: Six Sigma, TQM
  - We definitely need to teach key software engineering quality processes throughout ICT curricula:
    - Testing
    - Validation and verification

- Product quality
  - Use software engineering failures as teaching tools
    - This is done much more in other engineering disciplines.
    - Software engineering has good examples of failed development projects and software processes; all ICT students need to see such examples.
    - Are there examples of poorly crafted industrial-strength software products below the system level: algorithms, requirements, architecture, design, code?
    - Such examples would be very useful in the classroom, but they may be difficult to obtain.

- What about artifacts from other ICT disciplines? Do we have examples of industrial-strength instruction sets of poor quality that can be used in teaching?
- What about user interfaces, hardware architectures, and operating system designs?

- We need to work with beginning students to get them to understand that quality is an essential aspect of good code.
- We need to reinforce this understanding throughout all ICT curricula for all artifacts.